Application of Neural Network Learning Algorithms on the 'NSL_KDD.csv' Dataset.

FAIT PAR:

ZAKARIA ELOFIR - CYBERSECURITY

PROFESSEUR

SAID HRAOUI

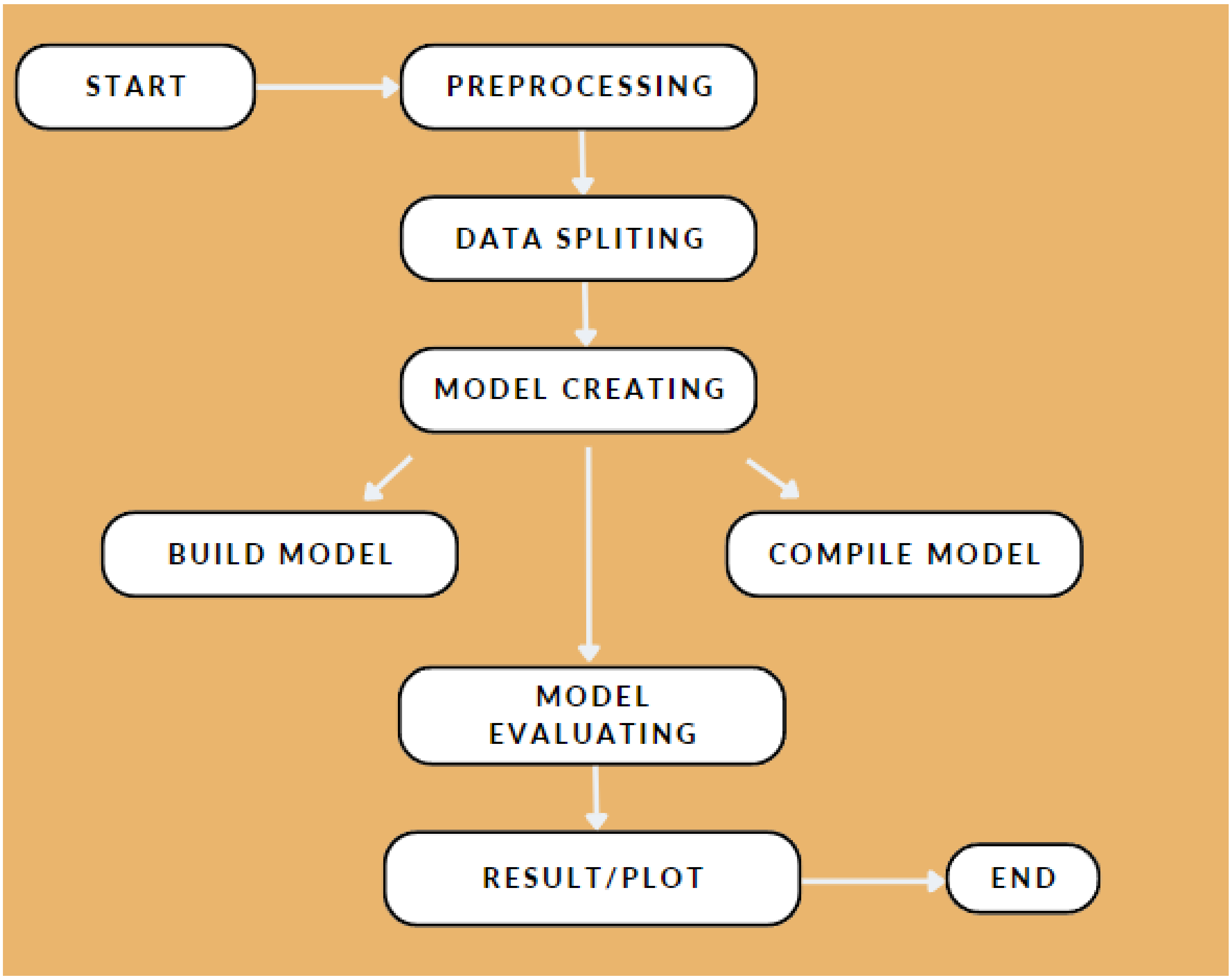
Introduction

Neural networks are powerful machine learning models widely used for classification and prediction in various fields. The objective of this assignment is to apply neural network learning algorithms on the 'NSL_KDD.csv' Dataset. This dataset contains data related to computer networks and is often used to assess the performance of intrusion detection models.

The provided code aimed at building and evaluating a deep learning model for network intrusion detection using the NSL-KDD dataset. The script uses various machine learning and data preprocessing libraries, such as NumPy, Pandas, Matplotlib, Seaborn, and Scikit-learn, as well as TensorFlow and Keras for building the deep learning model.

diagram

flowchart describing the typical steps involved in a machine learning project pipeline, specifically using neural networks.



Data Loading and Exploration:

The script begins by importing necessary libraries and loading the NSL-KDD dataset from a CSV file. The dataset contains information about network connections and their labels, indicating whether they are normal or represent various types of network attacks.

The dataset is explored using basic Pandas functions:

- Checking unique labels in the dataset.
- Displaying the data type of each feature.
- Checking for any missing values in the dataset.

The output of data.info() for the NSL-KDD dataset provides the following insights:

1. Total Entries: The dataset contains 148,517 entries (rows).
2. Total Features: There are 42 columns (features) in the dataset.
3. Feature Details:
   - Integer Features: 23 features are of integer type (int64). These are likely to be counts, flags (binary values), or other numerical measures.
   - Float Features: 15 features are of floating-point type (float64). These could represent rates or proportions.
   - Object Features: 4 features are of object type, typically representing categorical data. These include protocol_type, service, flag, and label.
4. Non-Null Counts: All features have 148,517 non-null entries, indicating that there are no missing values in any of the columns.

Data Preprocessing:

1. Correlation Analysis:

- A correlation matrix is generated using Seaborn to identify the relationships between different features.
- The 'num_outbound_cmds' column is dropped from the dataset.

2. Label Merging:

Labels are reorganized into broader categories (normal, probing, DoS, U2R, R2L) for classification purposes.

3. Data Encoding:

Categorical features ('protocol_type', 'service', 'flag') are encoded using LabelEncoder.
Dummy encoding is applied to the 'label' column using one-hot encoding.

4. Feature Selection:

Relevant features are selected for model training based on the correlation analysis.

5. Data Splitting and Scaling:

The dataset is split into training and testing sets.
RobustScaler and StandardScaler are applied to scale the features.

selecting imporatnat features

```
Entrée [23]: ImportantFeatures=data.columns[0:40]
             ImportantFeatures
```

```
   Out[23]: Index(['duration', 'protocol_type', 'service', 'flag', 'src_bytes',
                   'dst_bytes', 'land', 'wrong_fragment', 'urgent', 'hot',
                   'num_failed_logins', 'logged_in', 'num_compromised', 'root_shell',
                   'su_attempted', 'num_root', 'num_file_creations', 'num_shells',
                   'num_access_files', 'is_host_login', 'is_guest_login', 'count',
                   'srv_count', 'serror_rate', 'srv_serror_rate', 'rerror_rate',
                   'srv_rerror_rate', 'same_srv_rate', 'diff_srv_rate',
                   'srv_diff_host_rate', 'dst_host_count', 'dst_host_srv_count',
                   'dst_host_same_srv_rate', 'dst_host_diff_srv_rate',
                   'dst_host_same_src_port_rate', 'dst_host_srv_diff_host_rate',
                   'dst_host_serror_rate', 'dst_host_srv_serror_rate',
                   'dst_host_rerror_rate', 'dst_host_srv_rerror_rate'],
                  dtype='object')
```

Splitting the labels and features for making data for model

```
Entrée [27]: x= data[ImportantFeatures].values
             y= data[['label_normal', 'label_Probing', 'label_Dos', 'label_U2R', 'label_R2L']].values
```

Split data into train and test

```
Entrée [28]: from sklearn.model_selection import train_test_split
             X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.33, random_state=42
```

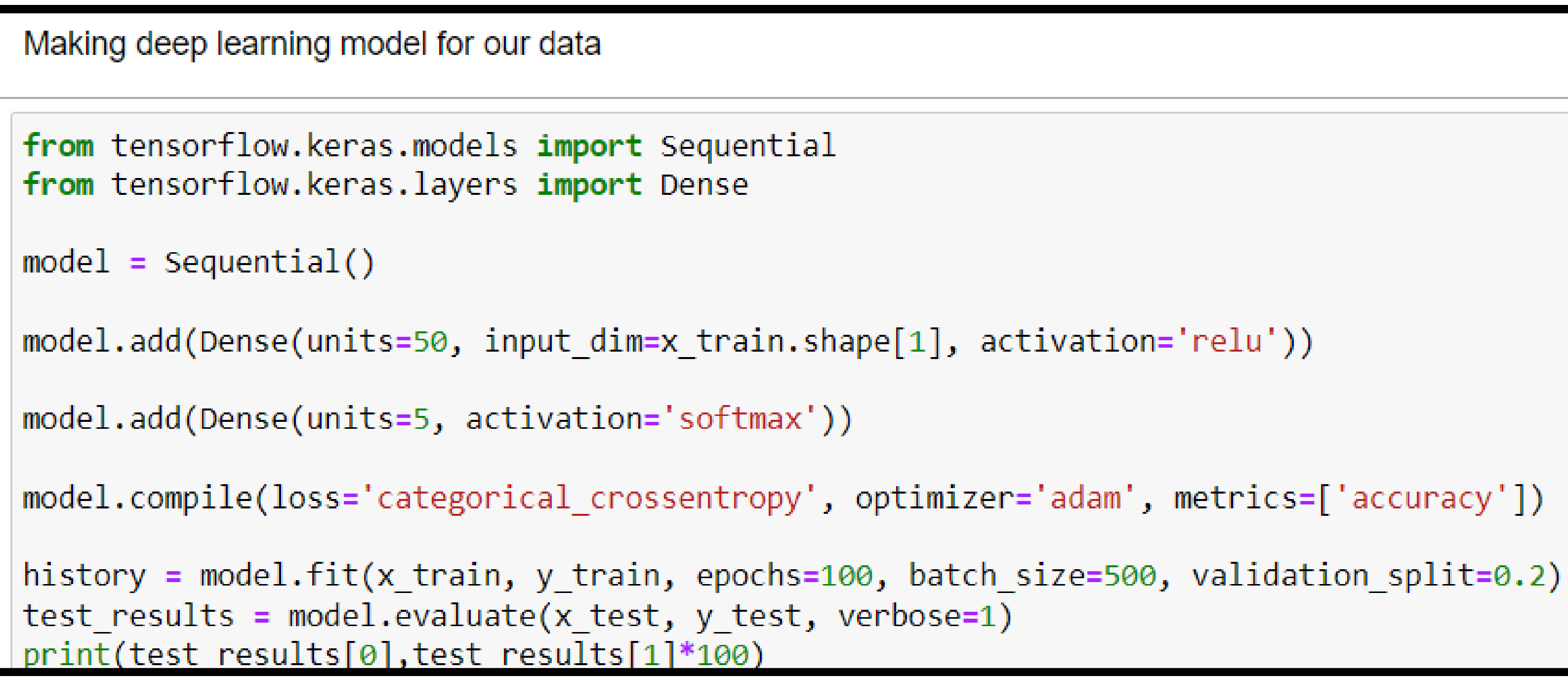
Scale data to have good model

```
Entrée [30]: ro_scaler = RobustScaler()
             x_train = ro_scaler.fit_transform(X_train)
             x_test = ro_scaler.transform(X_test)
```

```
Entrée [31]: ros_scaler = StandardScaler()
             x_train = ros_scaler.fit_transform(x_train)
             x_test = ros_scaler.transform(x_test)
```

Model Creation:

- A deep learning model is created using Keras Sequential API.
- The model has an input layer with 50 units and a hidden layer with ReLU activation.
- The output layer has 5 units with a softmax activation function to predict the different attack classes.
- Categorical cross-entropy is used as the loss function, and Adam optimizer is employed for model compilation.



Making deep learning model for our data

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

model = Sequential()

model.add(Dense(units=50, input_dim=x_train.shape[1], activation='relu'))

model.add(Dense(units=5, activation='softmax'))

model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

history = model.fit(x_train, y_train, epochs=100, batch_size=500, validation_split=0.2)
test_results = model.evaluate(x_test, y_test, verbose=1)
print(test_results[0],test_results[1]*100)
```

- Model Architecture: The model is relatively simple and may be appropriate for a dataset with a moderate number of features and classes.

- Training Process: The model was trained for 100 epochs with a batch size of 500. The training process includes a validation split of 20%, which is used to monitor and prevent overfitting.

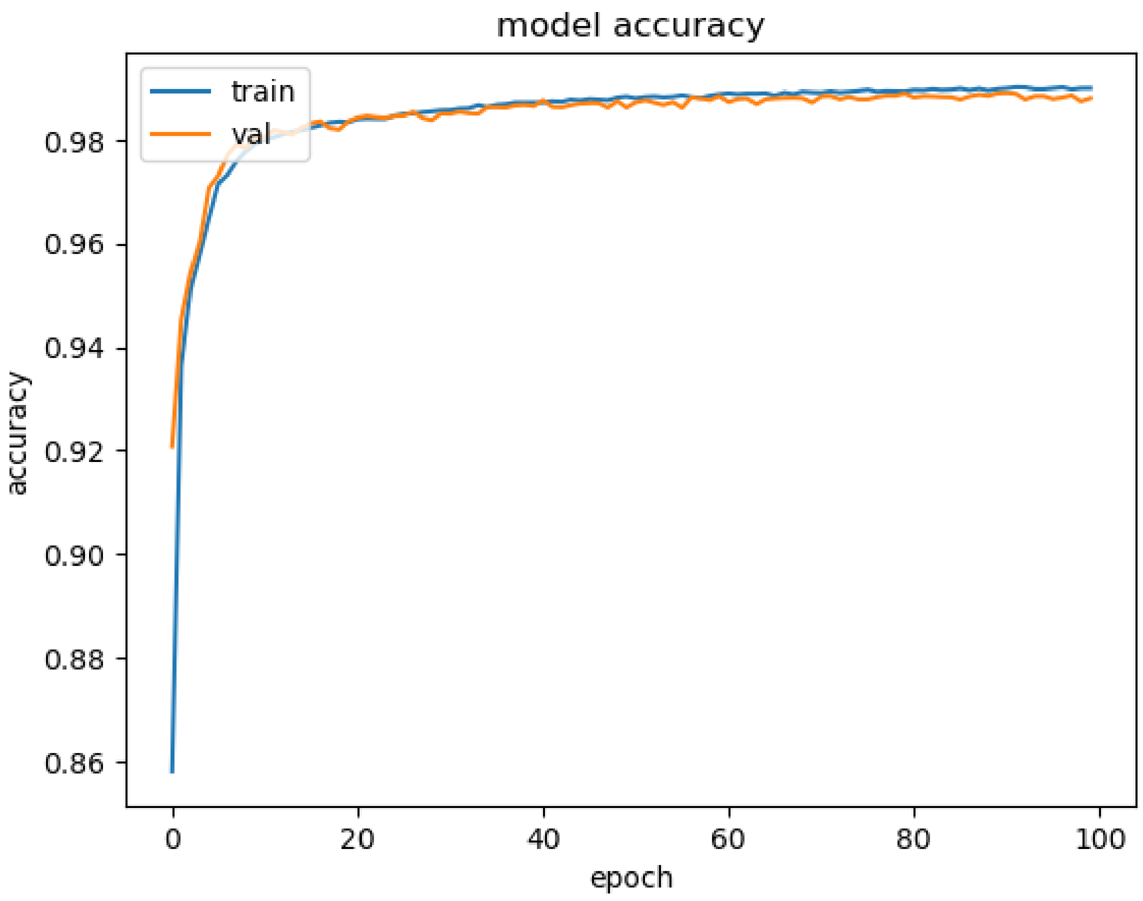
- Performance Improvement: Over the epochs, both training and validation accuracy improve, which is a good sign that the model is learning. The loss decreases, indicating better model performance over time.

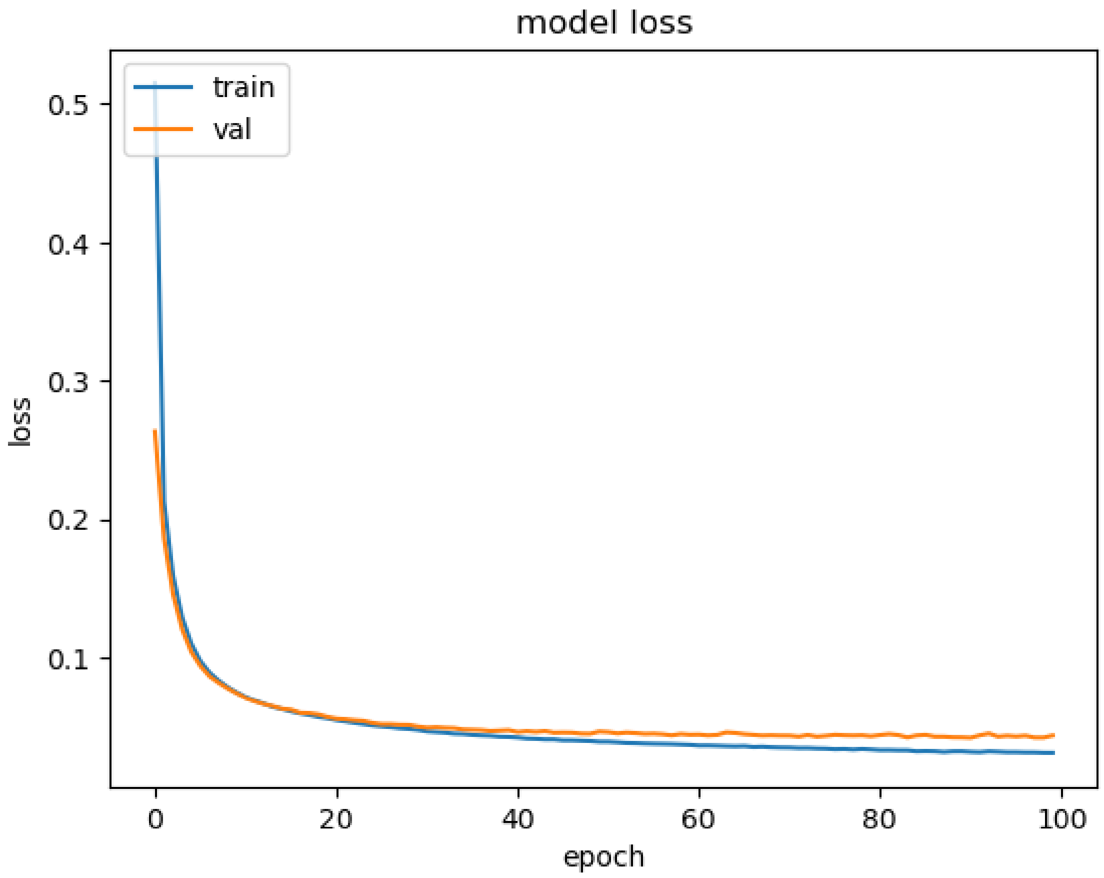
- Final Evaluation: After training, the model achieves a test accuracy of approximately 98.73%, with a loss of 0.0457. This is a high level of accuracy and suggests that the model performs well on the test set.

- Validation Performance: The validation accuracy is consistent with the test accuracy, suggesting that the model generalizes well and is not overfitted to the training data.

Model Training:

The model is trained for 100 epochs with a batch size of 500 and a validation split of 20%.





the training history of the neural network model in terms of accuracy and loss over 100 epochs. Here are some observations from each:

Model Accuracy Plot Observations:

- Rapid Learning: There's a sharp increase in accuracy for both training and validation sets in the initial epochs, which suggests that the model is learning rapidly early on.

- Convergence: The training and validation accuracy converge closely after the initial epochs, which indicates that the model generalizes well to unseen data.

- Stable Accuracy: After around 20 epochs, both the training and validation accuracy lines flatten out, suggesting little to no improvement in accuracy with further training.

- High Accuracy: The model achieves high accuracy (close to 98-99%) on both the training and validation sets, which is excellent for most classification tasks.

Model Loss Plot Observations:

- Steep Initial Decline: Similar to the accuracy plot, there's a steep decline in loss for the training and validation sets during the initial epochs.
- Loss Stabilization: The loss stabilizes after the sharp initial decline, with minimal improvements after around 20 epochs.
- Low Loss Values: The final loss values are low, indicating a good fit. There is no significant divergence between training and validation loss, which would have suggested overfitting.
- Consistency Between Sets: The loss for the training and validation sets is consistent throughout the training process, further indicating that the model is generalizing well.

Overall, these plots suggest that the neural network model trained on the dataset is performing well with a stable learning process. It achieves high accuracy and low loss, indicating a successful learning phase.

Model Evaluation:

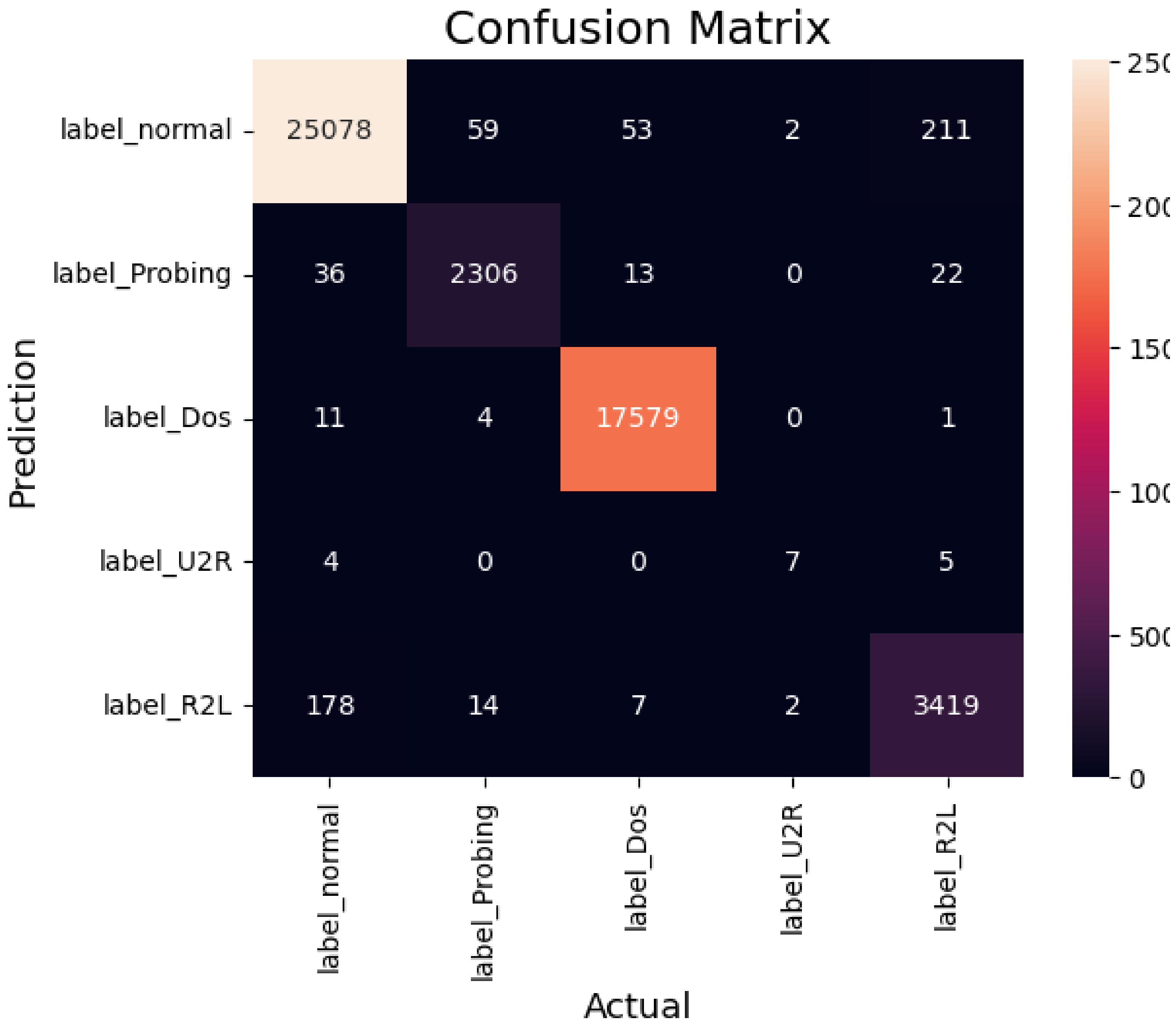
The trained model is evaluated on the test set, and accuracy is printed.

Confusion Matrix:

A confusion matrix is generated to visualize the model's performance on the test set.

Classification Report:

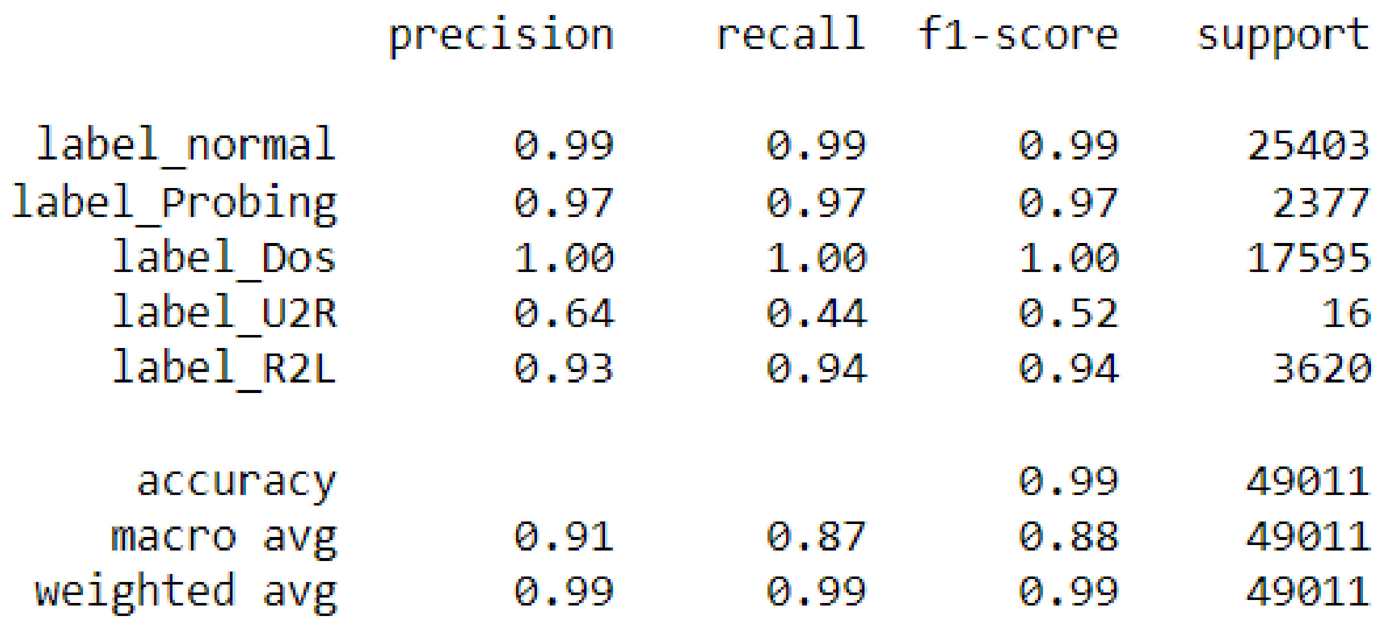
A classification report is printed, providing precision, recall, and F1-score for each class.



classification report:

- For the "label_normal" class, the model achieved high precision, recall, and F1-score, indicating excellent performance for this class.
- The "label_Dos" class also has high precision, recall, and F1-score, suggesting that the model performs very well for this class.
- The "label_Probing" class has good precision, recall, and F1-score, indicating a decent performance.
- The "label_R2L" class also shows good precision, recall, and F1-score, suggesting good performance.
- However, the "label_U2R" class has lower precision and recall, which might indicate that the model struggles to correctly classify instances in this class.

Overall, the model appears to perform well, with high accuracy and good scores for most classes. could be improved by further optimization and fine tununings



```
               precision    recall  f1-score   support

 label_normal       0.99      0.99      0.99     25403
label_Probing       0.97      0.97      0.97      2377
    label_Dos       1.00      1.00      1.00     17595
    label_U2R       0.64      0.44      0.52        16
    label_R2L       0.93      0.94      0.94      3620

     accuracy                           0.99     49011
    macro avg       0.91      0.87      0.88     49011
 weighted avg       0.99      0.99      0.99     49011
```

FIN